Toy example with exact transitions

If u is an input of interest and v are the other inputs, recall that we compute the APC by sampling twice from u conditional on v, and average over the distribution of v (equation (5) in the APC paper defines the quantity we wish to approximate). So we're interested in the distribution of u given v.

If there were enough pairs of points with identical v, we could just use the sample distribution of u given v. As noted in the paper, we may have few (if any) pairs of points with identical v. But still, it's worth thinking through an example where we do.

Suppose v consists of only 1 input, which can either be $v = v_1$ or $v = v_2$. For simplicity, assume u only has exactly two possible (equally likely) values at each v, so there is only one possible transition at each v. Here's an example:

```
exampleDF <- data.frame(
  v=c(3,3,7,7),
  u=c(10,20,12,22)
  )[rep(c(1,2,3,4),c(40,40,10,10)),]

# Count each u/v combination:
```

```
## |  v|   u|  CountOfRows|
## |--:|---:|------------:|
## |  3|  10|           40|
## |  3|  20|           40|
## |  7|  12|           10|
## |  7|  22|           10|
```

Say we have a model $\hat{y} = f(u, v)$. I'll choose $\hat{y} = f(u, v) = uv$ for a simple example. (How the model is estimated is completely orthogonal to the questions addressed here.)

Equation (2) in the paper says the numerator in the APC should be:

$$(.4)(.5)(.5)(f(20, 3) - f(10, 3)) + (0.1)(.5)(.5)(f(22, 7) - f(12, 7))$$

The .5's are the $p(u|v)$'s (and will cancel out in this case). (Terms with transition size 0 aren't included.)

The denominator is:

$$(.4)(.5)(.5)((20 - 10) + (.1)(.5)(.5)((22 - 12)$$

The ratio simplifies to:

$$.8\delta_u(10 \to 20, 3, f) + 0.2\delta_u(12 \to 22, 7, f)$$

This is all overkill for our very simple example, where it's easy to see that the APC is just $(.8)(3) + (.2)(6)$. But I wanted to be very concrete.

I'll compute it:

```
f <- function(u, v) return(u*v)
ApcExact <- .8*(f(20,3) - f(10,3))/10 + .2*(f(22,7) - f(12,7))/10
ApcExact
```

```
## [1] 3.8
```

Now without exact duplicates

Now imagine we don't have any exact duplicates of v. To get a corresponding example like that, I'll add a really tiny bit of noise to v in the example, $v_{new} = v + N(0, \epsilon)$.

```
exampleDF2 <- transform(exampleDF, v = v + rnorm(nrow(exampleDF), sd=.001))
```

Now we form pairs and compute weights as described in the paper. Here's a sample of the resulting data frame of pairs:

```
pairsDF <- get_pairs(exampleDF2, u="u", v="v", renormalizeWeights=FALSE)
```

```
## |      v|   u|  originalRowNumber|    v.B|  u.B|  originalRowNumber.B|  weight|
## |------:|---:|------------------:|------:|----:|--------------------:|-------:|
## |  3.000|  20|                 53|  3.000|   20|                   57|  1.0000|
## |  3.000|  10|                 34|  3.000|   10|                   16|  1.0000|
## |  3.000|  10|                  3|  3.001|   10|                   21|  1.0000|
## |  3.000|  20|                 63|  3.000|   20|                   41|  1.0000|
## |  7.001|  12|                 87|  3.000|   20|                   62|  0.1391|
## |  2.999|  10|                 19|  3.000|   20|                   48|  1.0000|
## |  2.999|  20|                 58|  7.000|   12|                   90|  0.1390|
## |  3.000|  10|                 39|  3.000|   20|                   60|  1.0000|
## |  3.001|  10|                  1|  7.000|   22|                   96|  0.1392|
## |  3.000|  20|                 44|  2.997|   20|                   75|  1.0000|
## |  3.001|  10|                 25|  2.999|   20|                   73|  1.0000|
## |  3.002|  10|                 13|  3.000|   20|                   52|  1.0000|
```

Now pairs with nearby v's (which would have been the same v's previously) have high weights, where pairs from far-away v's (which were different v's in the previous example) have low weights. That's good.

But we have a problem, which is that v near 3 now has more weight in the data set for two reasons:

1. we started with more v's near 3, so there are more rows with v near 3 as the first element of the pair; and
2. each time v is near 3 in the first element of each pair, there are more nearby v's to pair with, so we get higher weights.

Reason (1) is good, but reason (2) is not so good.

In the data frame of pairs, the weights are all close to 0.14 or 1. Let's look at how the distribution of u and v in just the pairs with weights close to 1:

```
pairsDF <- data.frame(vRounded = round(pairsDF$v), pairsDF)
pairsHighWeightsDF <- subset(pairsDF, weight > 0.9)
ddply(pairsHighWeightsDF,
      c("vRounded","u"),
      function(df) data.frame(CountOfRows = nrow(df),
                              ProportionOfRows = nrow(df)/nrow(pairsHighWeightsDF)))
```

```
##   vRounded  u CountOfRows ProportionOfRows
## 1        3 10        3160          0.47164
## 2        3 20        3160          0.47164
## 3        7 12         190          0.02836
## 4        7 22         190          0.02836
```

We see that v's near 7 makes up only about 5.7% of the pairs. (It would be exactly $(.2)(.2) = 4\%$, except that when we form pairs to compute the APC we don't pair any row with itself.)

If we form the APC based on these pairs and these weights, we weight the v's near 3 too much, so our APC is too low:

```
pairsDF$yHat1 <- f(pairsDF$u, pairsDF$v)
pairsDF$yHat2 <- f(pairsDF$u.B, pairsDF$v)
pairsDF$uDiff <- pairsDF$u.B - pairsDF$u
ApcApprox1 <-
  with(pairsDF,
       sum(weight * (yHat2 - yHat1) * sign(uDiff)) / sum(weight * uDiff * sign(uDiff)))
ApcApprox1
```

```
## [1] 3.364
```

I showed the computation above, but we can also use the `get_apc` function:

```
get_apc(function(df) return(df$u * df$v), exampleDF2, u="u", v="v", renormalizeWeights=FALSE
```

```
## [1] 3.364
```

Instead, we can normalize weights so that within each first element of the pair.

```
pairsDFWeightsNormalized <- ddply(pairsDF, "originalRowNumber", transform, weight = weight/s
ApcApprox2 <-
  with(pairsDFWeightsNormalized,
       sum(weight * (yHat2 - yHat1) * sign(uDiff)) / sum(weight * uDiff * sign(uDiff)))
ApcApprox2
```

```
## [1] 3.854
```

These renormalized weights are the ones returned from `get_pairs` by default,
and used in `get_apc` by default:

```
get_apc(function(df) return(df$u * df$v), exampleDF2, u="u", v="v")
```

```
## [1] 3.854
```